

(19)



Europäisches Patentamt  
European Patent Office  
Office européen des brevets



(11)

EP 1 199 629 A1

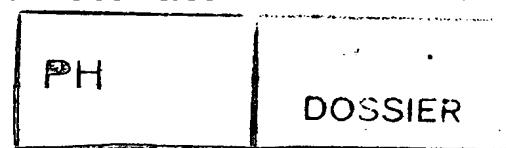
(12)

## EUROPEAN PATENT APPLICATION

(43) Date of publication:  
24.04.2002 Bulletin 2002/17

(51) Int Cl.7: G06F 9/38

(21) Application number: 00830673.0



(22) Date of filing: 17.10.2000

(84) Designated Contracting States:  
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU  
MC NL PT SE  
Designated Extension States:  
AL LT LV MK RO SI

- Silvano, Cristina, c/o Politecnico di Milano 20133 Milano (IT)
- Zaccaria, Vittorio, c/o Politecnico di Milano Via Ponzo 34/5 20133 Milano (IT)
- Pau, Danilo 20099 Sesto San Giovanni Milano (IT)
- Zafalon, Roberto 30135 Venezia (IT)

(71) Applicant: STMicroelectronics S.r.l.  
20041 Agrate Brianza (Milano) (IT)

(72) Inventors:  

- Sami, Mariagiovanna, c/o Politecnico di Milano 20133 Milano (IT)
- Sciuto, Donatella, c/o Politecnico di Milano 20133 Milano (IT)

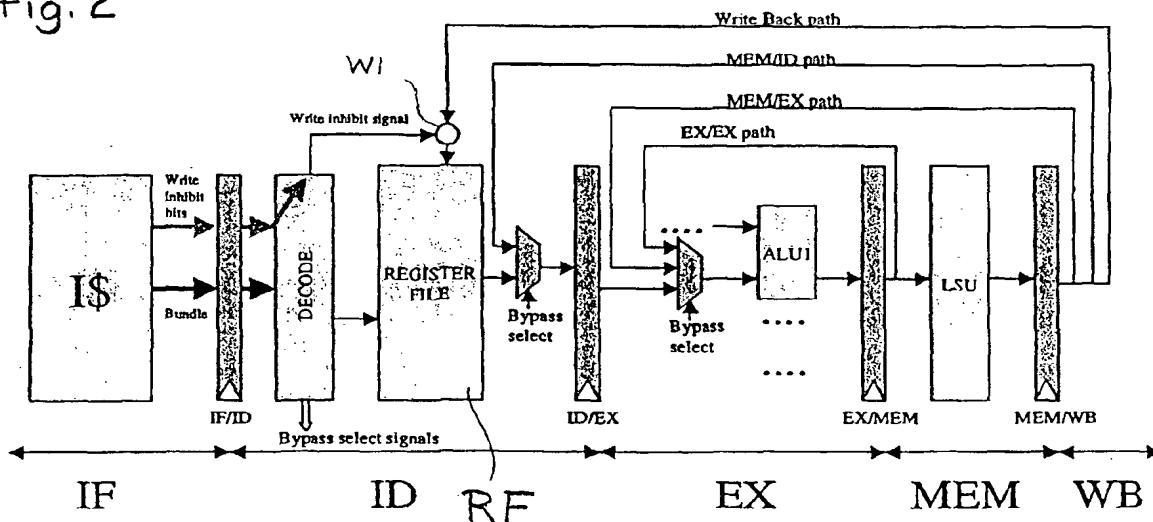
(74) Representative: Bosotti, Luciano et al  
c/o Buzzi, Notaro & Antonielli d'Oulx  
Via Maria Vittoria 18  
10123 Torino (IT)

### (54) Processor architecture with variable-stage pipeline

(57) An architecture for a pipeline processor circuit, preferably of the VLIW type, comprises a plurality of stages (IF, ID, EX, MEM, WB) and a network of forwarding paths (EX-EX, MEM-EX, MEM-ID) which connect pairs of said stages, as well as a register file (RF) for

operand write-back. An optimization-of-power-consumption function is provided via inhibition of writing (Write inhibit) and subsequent readings in said register file (RF) of operands retrievable from said forwarding network on account of their reduced liveness length.

Fig. 2



EP 1 199 629 A1

**Description**Scope of the invention

5 [0001] The present invention relates to processor architectures, in particular of the type currently referred to as "pipeline" architectures.

Description of prior art

10 [0002] One of the main effects of the introduction of the pipelining technique is the modification of the relative timing of instructions resulting from the overlapping of their execution, which introduces factors of conflict or hazard due both to data dependence (data hazards) and to modifications of the control stream (control hazards). In particular, such conflicts emerge when sending of instructions through the pipeline modifies the order of read/write accesses to operands with respect to the natural order of the program (i.e., with respect to the sequential execution of instructions in non-pipelined processors).

15 [0003] In this connection, useful reference may be made to J. Hennessy and D.A. Patterson, "Computer Architecture: A Quantitative Approach", Morgan Kaufmann Publishers, San Mateo, CA, Second Edition, 1996.

20 [0004] The set of problems linked in particular to data hazards may be solved at a hardware level with the technique currently referred to as "forwarding" (or also "bypassing", and sometimes "short-circuiting"). This technique uses the interstage registers of the pipeline architecture for forwarding the results of an instruction  $i_1$ , produced by one stage of the pipeline, directly to the inputs of the previous stages of the pipeline in order to be used in the execution of instructions that follow  $i_1$ . A result may therefore be forwarded from the output of one functional unit to the inputs of another unit that precedes it in the flow along the pipeline, and likewise starting from the output of one unit to the inputs of the same unit.

25 [0005] In order to ensure this forwarding mechanism, it is necessary to provide, in the processor, the required forwarding paths and the control of these paths. The forwarding technique may require a specific path starting from any register of the pipeline structure to the inputs of any functional unit, as in the case of the architecture known as "DLX", to which reference is made in the text cited previously.

30 [0006] Data bypassed to the functional units of the early pipeline stages are normally in any case stored in the register file (RF) during the last pipeline stage (i.e., the so-called "write-back stage") in view of a subsequent use in the program being executed. Processors that use the forwarding technique achieve substantial improvements in terms of performance owing to the elimination of stall cycles introduced by data-hazard factors.

35 [0007] The main problems linked to the forwarding mechanism in the sphere of processors, and in particular in the sphere of the so-called "very-long-instruction-word or VLIW processors" have been investigated in studies, such as A. Abnous and N. Bagherzadeh, "Pipelining and Bypassing in a VLIW Processor", IEEE Trans. on Parallel and Distributed Systems, Vol. 5, No. 6, June 1994, pp. 658-663, and H. Corporaal, "Microprocessor Architectures from VLIW to TTA", John Wiley and Sons, England.

[0008] The above works analyze the advantages in terms of performance of various bypassing schemes, in particular as regards their effectiveness in solving data hazards in both four-stage and five-stage pipeline architectures.

40 [0009] The idea of exploiting register values that are bypassed during pipeline stages has been combined with the introduction of a small register cache with the purpose of improving performance, as is described in the work by R. Yung and N.C. Wilhelm, "Caching Processor General Registers", ICCD '95. Proceedings of IEEE International Conference on Computer Design, 1995, pp. 307-312. In this architecture, referred to as "Register Scoreboard and Cache", pipeline operands are supplied either by the register cache or by the bypass network.

45 [0010] In the work by L.A. Lozano and G.R. Gao, "Exploiting Short-lived Variables in Superscalar Processors", MICRO-28, Proceedings of 28th Annual IEEE/ACM International Symposium on Microarchitecture, 1995, pp. 292-302, a scheme is proposed for superscalar processors which comprises an analysis carried out by the compiler and an extension of the architecture in order to avoid definitive writings in the RF (commits) of the values of variables which are bound to be short-lived and which, consequently, do not require long-term persistence in the RF. The advantages provided by this solution have been assessed by the authors prevalently in terms of reduction of the write ports to the RF and of reduction in the amount of transfers from registers to memory required, so as to achieve improvements in execution time. The work referred to reports the improvements linked to this solution in terms of performance, without any consideration, however, of the effects in terms of power absorption.

50 [0011] The concept of avoiding the presence of information without any useful value (dead-value information) in the RF is analyzed in the work by M.M. Martin, A. Roth, and C.N. Fischer, "Exploiting Dead Value Information", MICRO-30, Proceedings of 30th Annual IEEE/ACM International Symposium on Microarchitecture, 1997, pp. 125-135. The values in the registers are considered useless or "dead" when they are not read before being overwritten. The advantages of this solution have been studied in terms of reduction in RF size and elimination of unnecessary save/restore

instructions from the execution stream at procedure calls and across context switches.

**[0013]** The majority of low-power-absorption techniques developed for digital CMOS circuits aim at reducing switching power, which represents the most significant contribution to the global power budget. For high-performance processors, low-power-absorption solutions aim at reducing the effective capacitance  $C_{\text{EFF}}$  of the processor nodes being switched.

**[0014]** The parameter  $C_{\text{EFF}}$  of a node is defined as the product of the load capacitance  $C_L$  and the switching activity  $\alpha$  of the node. In digital CMOS processors it is possible to obtain considerable economy in terms of power absorption by minimizing the transition activity of high-capacitance buses, such as data-path buses and input/output buses. Another significant component of the power budget in modern processors is represented by multi-port RF accesses and other on-chip cache accesses.

## Object and summary of the present invention

20 [0015] The object of the present invention is to provide a processor architecture that is able to overcome the drawbacks and limitations outlined previously.

**[0016]** In accordance with the present invention, the above object is achieved thanks to an architecture having the characteristics specifically referred to in the ensuing claims.

[0017] In particular, the main object of the invention is to define a criterion for optimization of the architecture for static-scheduling pipelined processors, and in particular for VLIW-architecture pipelined processors capable of exploiting the data-forwarding technique in regard to short-lived variables, in order to cut down on power absorption.

**[0018]** Basically, the idea underlying the invention consists in reducing the RF-access activity by avoiding long-term storage of short-lived variables. This is possible - with a negligible overhead in hardware terms - thanks to the pre-existing availability of interstage registers and of appropriate forwarding paths. Short-lived variables are simply stored locally by the instruction that produces them in the interstage registers and are forwarded directly to the appropriate stage of the instruction that uses them, exploiting the forwarding paths. The instruction that produces the variables does not therefore carry out a costly action of write-back to the RF, and, in turn, the instruction that uses the variables does not have to perform any read operation from the RF.

**[0019]** The application of this technique entails evaluation of the liveness length  $L$  of the  $n$ -th assignment to a register  $R$ , defined as the distance between its  $n$ -th assignment and its last use. This information makes it possible to decide whether the variable is to be stored in the RF in view of a subsequent use, or whether its use is in fact limited to just a few clock cycles. In the latter case, the variable is short-lived, and its value may be passed on as an operand to the subsequent instructions, by using the forwarding paths, thus avoiding the need to write it in the RF.

**[0020]** The decision whether to enable the RF write phase may be taken by the hardware during execution, or else anticipated during compiling of the source program. Unlike what occurs in superscalar processors, where the majority of the decisions are taken by the hardware at the moment of execution, the application of the low-power-absorption bypass technique in VLIW architectures may be performed during static scheduling by the compiler. This procedural approach reduces the complexity of the processor control logic.

[0021] The proposed architecture becomes particularly attractive in the case of certain applications of the embedded type whereby the analysis of register liveness length has shown that the interval of re-use of more than half of all the register definitions is limited to the next two instructions.

**[0022]** The most important characteristics of the solution according to the invention are the following:

- the solution according to the invention proposes an extension of an architectural type of the processor bypass network so as to prevent writing and subsequent reading of short-lived variables to/from the RF;
  - it is possible to analyze the effects on the compiler of the low-power-absorption architecture solution proposed for VLIW processors, by showing the possible implementation to keep the hardware limited;
  - it is possible to handle exceptions (e.g., error traps, division by zero, etc.);
  - the solution according to the invention may be extended also to processors with more than five pipeline stages (comprising more than three forwarding paths) so as to cut down on power absorption for variables the liveness length of which is greater than three;
  - the solution according to the invention opens up the road to further economies in terms of power absorption, which may be obtained by an optimization of instruction scheduling, exploiting to the full the intrinsic parallelism of such

processors and aiming at minimizing the "mean life" (liveness length) of the variables.

#### Brief description of the annexed drawings

- 5 [0023] The invention will now be described, purely to provide a non-limiting example, with reference to the attached drawings, in which:
- Figure 1 is a diagram illustrating the results of an analysis conducted on the active life of the registers in a processor framework;
  - 10 - Figure 2 illustrates, in the form of a functional block diagram, the application of an architecture according to the invention in a processor framework;
  - Figure 3 is a diagram illustrating the modalities with which exceptions are handled in a processor framework, in accordance with the invention; and
  - 15 - Figures 4 and 5 illustrate in still greater detail the modalities of implementation of a solution according to the invention.

#### Description of the theoretical bases of the invention

- 20 [0024] Before proceeding to a detailed description of an example of embodiment of the invention, it is useful to refer, in what follows, to the results of a number of experimental analyses conducted for establishing the liveness length of variables in embedded-type applications.
- [0025] The specific purpose was to measure - in the execution phase - the percentage of register definitions in the application code that can be read directly by the forwarding network without being written in the RF.
- 25 [0026] The analysis referred to above was conducted employing, as an example, a set of currently used DSP algorithms written in C language and compiled using a 32-bit 4-way industrial VLIW compiler.
- [0027] The register liveness-length analysis may be performed either statically or dynamically.
- [0028] Static analysis consists in inspecting, in a static way, the assembler code generated by the compiler in the context of each basic block, so as to detect the liveness length of the registers.
- 30 [0029] Dynamic analysis consists in inspecting the execution traces of the assembler code, a procedure that provides more accurate profiling information as regards register read/write accesses.
- [0030] The results reported in what follows relate to the dynamic solution.
- [0031] Each benchmark program was appropriately instrumented at the assembler level with an automatic tool and then simulated, so as to keep track of the relevant information at each clock cycle, namely:

- 35 - register definitions;
- register uses; and
  - basic-block boundaries encountered.

- 40 [0032] For each basic block in the trace, analysis of register liveness length was performed by defining the liveness length  $L_n(R)$  of the  $n$ -th assignment to a register  $R$  as the distance (expressed in number of instructions) between the  $n$ -th assignment and its last use:

$$L_n(R) = U_n(R) - D_n(R),$$

45 where  $D_n(R)$  is the trace index of the instruction that made the  $n$ -th assignment to  $R$ , and  $U_n(R)$  is the index of the last instruction that used the  $n$ -th assignment to  $R$  prior to redefinition of  $R$  during the  $(n+1)$ -th assignment  $D_{n+1}(R)$ .

[0033] In a VLIW architecture it is possible to assume a throughput of one very long instruction per clock cycle.

50 [0034] In order to maintain the analysis extremely conservative, the computation of  $L_n(R)$  was performed applying the following restrictions:

- $U_n$  and  $D_n$  are in the same basic block; and
- $D_{n+1}$  and  $D_n$  are in the same basic block.

55 [0035] These rules enable a simplification of the analysis by considering only liveness ranges that do not transcend the boundaries between the basic blocks. However, this assumption does not constitute an important limitation, given that the majority of modern VLIW compilers maximize the size of basic blocks, so generating a relevant number of liveness ranges that are resolved completely within the respective basic block.

**[0036]** To clarify the above concept, we can analyze an assembler-code trace for a 4-way VLIW machine which executes a discrete-cosine-transform (DCT) algorithm. The code analyzed is made up of four very long instructions (namely, 27268, 27269, 27270, and 27271):

```

5   27268  shr $r16 = $r16, 8
    sub $r18 = $r18, $r7
    add $r17 = $r17, $r19
    sub $r19 = $r19, $r15;

10  27269  shr $r18 = $r18, 8
    shr $r17 = $r17, 8
    shr $r19 = $r19, 8
    mul $r20 = $r20, 181;

15  27270  shr $r10 = $r10, $r8
    mul $r11 = $r11, 3784
    sub $r5 = $r12, $r9;

20  27271  sub $r10 = $r10, $r3
    add $r20 = $r20, 128
    brf $r26, label_232;

```

in which each very long instruction is identified by an execution index, by a set comprising from one to four operations, and terminates with a semicolon.

**[0037]** In the above example, a boundary may be noted which concludes a basic block at the instruction 27271 (the conditional-branch operation).

**[0038]** If we consider the liveness of the assignment of the register \$r18 in 27268 ( $D_n$ ), it may seen that this definition is used for the last time in 27269, given that there is another definition of the register \$r18 in the same cycle (namely,  $D_{n+1}$ ). The value of  $L_n$  of \$r18 is therefore equal to one clock cycle. It should be noted that it is not possible to compute the value  $L_{n+1}$  of \$r18, since there are neither last uses  $U_{n+1}$  nor redefinitions  $D_{n+2}$  in the same basic block.

**[0039]** For the purposes of the analysis, a set of test programs (benchmark set) was selected made up of the following algorithms:

- a finite-impulse-response (FIR) filter;
- a sample program performing a discrete cosine transform (DCT) and an inverse discrete cosine transform (IDCT);
- an optimized DCT;
- an optimized IDCT; and
- a wavelet transform.

**[0040]** Note that, in order to improve performance, the optimized versions of the DCT/IDCT algorithms are characterized by a lower number of accesses to memory and a higher re-use of the registers as compared to the other algorithms.

**[0041]** The distribution of the register-liveness values detected by the algorithms considered is given in Table 1 below and is summarized graphically in Figure 1 of the attached drawings.

Algorithm	Register Liveness Length (in clock cycles)							
	1	2	3	4	5	6	7	8
FIR	0%	13%	10%	10%	0%	0%	0%	0%
DCT/IDCT	28%	12%	8%	3%	2%	1%	1%	0%
DCT (opt.)	32%	14%	11%	6%	2%	1%	0%	0%
IDCT (opt.)	42%	12%	6%	5%	2%	1%	1%	1%
Wavelet	7%	17%	1%	0%	2%	0%	0%	0%

**[0042]** In the above table, the columns represent the percentage of the registers the liveness of which is equal to a given value  $L$  lying in the range from 1 to 8 clock cycles (instructions).

**[0043]** In Figure 1 of the attached drawings, given on the ordinate are the above percentage values as a function of

the values of L appearing on the abscissa.

**[0044]** Both from Table 1 and from Figure 1 it emerges that - albeit with simplifying hypotheses - for the optimized algorithms, approximately one half of all the register definitions have liveness values of not greater than two clock cycles (46% and 54% for the DCT algorithm and the IDCT algorithm, respectively) . On average, in 35.4% of the cases the distance between the definition of the register and its last use is less than or equal to two clock cycles, whereas in 42.6% the distance is less than or equal to three clock cycles.

**[0045]** The above analysis moreover does not take into account the case where a register is never read between two successive definitions. In actual fact, there may be an overwriting of the register, for instance across basic blocks or during processor context switches (e.g., in response to an external interruption), but this phenomenon cannot be estimated in a static way at compiling in the framework of a basic block. Albeit advantageous for the solution according to the invention, the phenomenon is, however, not relevant for the current analysis, which focuses on an optimization function applicable in the framework of a basic block during the VLIW static compiling phase.

#### Detailed description of a preferred embodiment of the invention

**[0046]** Purely to provide a non-limiting example, the diagram of Figure 2 refers to a 4-way VLIW processor architecture with a 5-stage pipeline provided with forwarding logic.

**[0047]** The pipeline stages are the following:

- IF: instruction fetch from the instruction cache (I-cache) ;
- ID: instruction decoding and operand reading from the RF;
- EX: execution of instructions in arithmetic logic units (ALUs) having a latency corresponding to one clock cycle;
- MEM: accesses to memory for load/store instructions; and
- WB: write-back of operands in the RF.

**[0048]** Three forwarding paths (EX-EX, MEM-EX, and MEM-ID) provide direct connections between pairs of stages through the EX/MEN and MEM/WB interstage registers.

**[0049]** The various symbols and designations given in Figure 2 are well known to persons skilled in the sector, and consequently do not call for a detailed description herein. This applies both in regard to their meaning and in regard to their function.

**[0050]** The architecture in question is applicable, for example, in embedded VLIW cores of the Lx family, jointly developed by Hewlett-Packard Laboratories and by the present applicant. Each cluster of the Lx family comprises four ALUs for 32-bit integer operands, two 16x32 multipliers, and one load/store unit. The RF comprises sixty-four 32-bit general-purpose registers and eight 1-bit branching registers.

**[0051]** With reference to the aforementioned forwarding network, consider a sequence  $W = w_1 \dots w_2 \dots w_n$  of very long instructions. A generic instruction  $w_k$  can read its operands from the following instructions:

- $w_{k-1}$  through the EX/EX forwarding path (used when  $w_k$  is in the EX stage);
- $w_{k-2}$  through the MEM/EX forwarding path (used when  $w_k$  is in the EX stage);
- $w_{k-3}$  through the MEM/ID forwarding path (used when  $w_k$  is in the ID stage) ;
- $w_{k-n}$  when  $n > 3$  through the RF.

**[0052]** As indicated, the solution according to the invention inhibits the writing and subsequent readings of the operands in the RF whenever the values written may be retrieved from the bypass network on account of their short liveness.

**[0053]** This occurs specifically through the *Write-Inhibit* signal which is generated selectively in the ID stage and is destined to act on a WI node interposed in the path of the *Write-Back* signal from the WB stage to the RF.

**[0054]** Assuming, for example, that an instruction  $w_d$  assigns a register R, the liveness length of which is less than or equal to 3, and that  $w_k$  uses R during this live interval, the basic idea is to reduce power absorption by:

- disabling writing of R in the WB stage of  $w_d$ ; and
- inhibiting  $w_k$  from asserting the RF read address to read R (retrieved from the bypass network).

**[0055]** In general, whereas avoidance of write-back must be explicitly indicated in the very long instruction  $w_d$ , the information regarding the need for the source operands to be derived from the forwarding paths is in any case made available by the control logic, whatever the liveness of the variable might be. Consequently, it is possible to avoid reading from the RF whenever the source operands are expected to be extracted from the forwarding paths.

**[0056]** The power-absorption optimization function described above is implemented by a dedicated logic in the ID

stage which disables the write-enable signals for writing in the RF and minimizes RF read-port switching activity by maintaining the input read addresses equal to those of the last useful-access cycle.

**[0057]** As a practical example, reference may be made to the sequence of instructions considered previously, and in particular to the instructions 27268 and 27269. Writing-back of the registers \$r18, \$r17 and \$r19 in the RF during execution of 27268 may be avoided, and the subsequent reading of these values during execution of 27269 may be carried out directly from the EX-EX path of the bypassing network.

**[0058]** In a superscalar processor, this behaviour should be controlled by hardware, analyzing the instruction window to compute register liveness and generate control signals to the pipeline stages.

**[0059]** In a VLIW architecture, all scheduling decisions concerning data, resources and control are solved during compiling in the code-scheduling phase, as described, for example, in A.V. Aho, R. Sethi, and J.D. Ullman, "Compilers: Principles, Techniques, and Tools", Addison-Wesley, 1986.

**[0060]** Consequently, the decision as to whether the destination register must be write-inhibited or not can be delegated to the compiler, thus limiting the hardware overhead.

**[0061]** To pass the information from the compiler to the hardware control logic, two different approaches may be adopted:

- reserving specific operation bits in the encoding of the very-long-instruction format; this is suitable during definition of the instruction set, but it may entail a slight increase in instruction-encoding length;
- exploiting unused instruction-encoding bits; this solution is suitable when the instruction set has already been defined: it affords the possibility of saving on instruction length, but at the possible expense of limiting power saving to a subset of the operations present in the instruction set.

**[0062]** In either case, whilst the RF switching activity is minimized, there is a slight increase in the switching activity of the memory units used to store instructions.

**[0063]** As far as the problem of exception handling is concerned, the state of the processor may be assumed as being one of the following:

- a permanent architectural state stored in the RF;
- a volatile architectural state stored in the pipeline interstage registers from which the forwarding network transfers the source operands.

**[0064]** The volatile architectural state is handled as a FIFO memory having a depth equal to the number of stages during which the result of an operation can be stored in the pipeline (in the case of the 5-stage pipeline architecture represented in Figure 2, this depth is equal to three).

**[0065]** In general, a pipelined processor ensures that, when an element exits the volatile state, it is automatically written-back in the RF.

**[0066]** Instead, in the solution described herein, when an element exits the volatile state and is no longer used, it can be discarded, so avoiding write-back in the RF. This behaviour can create some problems when an exception occurs during processing.

**[0067]** In the architecture proposed herein as a reference example, an exception may occur in particular during the ID, EX or MEM stages, and can be serviced in the WB stage.

**[0068]** According to the exception taxonomy defined in the work by H. Corporaal cited previously, it is assumed that the processor adopts the operating mode currently referred to as "user-recoverable precise mode".

**[0069]** According to this model, the exceptions may be either exact or inexact.

**[0070]** An exact exception caused by an instruction issued at time t is a precise exception such as to require that the state changes caused by handling the exception should be visible to all instructions issued at and after time t and to none of the instructions issued before. Furthermore, all state changes in instructions issued before time t are visible to the exception-handling function.

**[0071]** If it is assumed that exceptions are handled in exact mode, when the excepting instruction reaches the WB stage, the instructions in the pipeline are flushed and re-executed.

**[0072]** Consider the situation illustrated in Figure 3, where at cycle x an instruction  $w_k$  reads its values from a write-inhibited instruction  $w_{k-2}$  through the forwarding network. At the same time assume that the instruction  $w_{k-1}$  generates an exception during the MEM stage. The results of  $w_{k-2}$  would be lost, but it is necessary for these values to be used during re-execution of  $w_k$ . Since neither the forwarding network nor the RF contain the results of  $w_{k-2}$ , the architectural state seen during the re-execution of  $w_k$  (at cycle  $x + nn$ ) would be incorrect.

**[0073]** In order to guarantee that the instructions in the pipeline are re-executed in the correct processor state, the write-inhibited values must be written in the RF whenever an exception signal is generated in the ID, EX or MEM stages.

**[0074]** In the case of the previous example, namely with  $w_{k-1}$  generating an exception in the MEM stage, the solution

here described forces write-back of the results of  $w_{k-1}$  and  $w_{k-2}$  in the RF, so that during re-execution of  $w_k$  at cycle  $x + nn$  the operands are read from the RF.

**[0075]** If, instead, it is assumed that exceptions are handled in non-exact or "inexact mode", when an exception occurs the instructions present in the pipeline are executed until completion, without the effects of the exception that is serviced subsequently being seen. In this case, all instructions in the pipeline are forced to write back the results in the RF.

**[0076]** The architecture represented in Figure 2 is able to guarantee both of the exception-handling mechanisms described previously.

**[0077]** When the exceptions are handled in exact mode, the supported register liveness is less than or equal to two clock cycles (through the EX/EX and MEM/EX paths).

**[0078]** When the exceptions are handled in non-exact mode, the exploiting register liveness can be extended to three clock cycles (through the EX/EX, MEM/EX and MEM/ID paths).

**[0079]** With specific regard to the case of interrupts or "cache-miss" phenomena, the asynchronous nature of interrupts enables them to be handled as inexact exceptions by forcing each very long instruction in the pipeline to write back the results before handling the interrupt. Cache misses, instead, produce phenomena that can be likened to bubbles flowing through the pipeline; therefore, whenever a miss signal is raised by the cache control logic, write-back of the results of the instructions is forced in the pipeline.

**[0080]** For a further clarification of the foregoing description it may be noted that, according to one of the elements of major interest of the invention, data sections of interstage registers of the pipeline structure in practice become a further, higher-level, layer in the memory hierarchy.

**[0081]** Hereinafter these registers will be referred to as "microregisters".

**[0082]** Microregisters are visible to the compiler, but not to the programmer.

**[0083]** The optimization rules for their use are particular, and different from those of the elements of the RF.

**[0084]** Microregisters are not write-addressable (or rather, they are implicitly addressed), and the rules for read addressing are architecture-related, in so far as they are more restrictive than for RF elements.

**[0085]** As has been pointed out, the solution according to the invention is essentially based on the forwarding (or bypassing) function so as to avoid writing and reading in the RF in order to reduce power consumption.

**[0086]** Whenever the compiler identifies short-lived variables such as to render the use of forwarding possible, after it has verified that the conditions specified in what follows are satisfied it does not reserve registers in the RF for such variables.

**[0087]** As far as use by the compiler is concerned, the RF space is thus effectively increased, and hence register spilling and the resulting cache traffic are reduced.

**[0088]** Consider in detail the five-stage pipeline structure schematically represented in Figure 4, which as a whole is similar to the one represented in Figure 2, with the provision, however, of two stages, EX1 and EX2.

**[0089]** Take as example the following high-level language instruction:

$$x := a * b + c - d$$

which is translated into intermediate code as:

$$\begin{aligned} t_0 &= a * b \\ t_1 &= c - d \\ x &= t_0 + t_1 \end{aligned}$$

**[0090]** Assume an operation latency of 1 for the subtraction and 2 for the multiplication.

**[0091]** Denoting by  $\mu_0$  the result section in the latch at exit from the EX2 stage and by  $\mu_1$  the corresponding section in the latch at exit from the EX1 stage, the above three elementary operations translate into a pseudo-assembler language which exploits the microregisters as follows:

mul  $\mu_2$ , R1, R2    it is assumed that a, b, c, d are initially stored in the registers R1 to R4  
 sub  $\mu_1$ , R3, R4  
 add R5,  $\mu_1$ ,  $\mu_2$     the final result is stored in R5

and the forwarding paths from the latches are exploited as represented in Figure 4.

**[0092]** For a five-stage pipeline, the maximum allowable distance between writing a variable in a microregister and using the same variable is 3. This creates a constraint for the compiler, which is able to exploit microregisters only in so far as a scheduling within the acceptable distance is possible. Obviously, if deeper pipelines are adopted, greater

distances can be used (together with more complex scheduling procedures and further reduction in RF use).

[0093] This first example refers to a sequential code. In the case of cycles (loops), microregisters can be exploited across the loop boundaries as well, provided that the constraints outlined above can be satisfied both between the loops (inter-loop) as well as within the loops (intra-loop).

[0094] If extension to a simple (pure) VLIW architecture is now considered, the point of interest is represented by the possibility of there being syllables (in parallel in a single very long instruction) characterized by different latencies.

[0095] In this case, transfers between microregisters along the pipeline lanes may have to be taken into account.

[0096] Consider again the same code segment as above, and a two-lane VLIW architecture with one ALU and one multiplier, i.e., a structure corresponding to the one represented in Figure 5.

[0097] Assume moreover that the latencies are the same as above. The code is then scheduled as follows:

i1 mul  $\mu_0^2$ , R1, R2; sub  $\mu_1^1$ , R3, R4     the superscript denotes the stage; the subscript denotes the lane  
 i2 nop     the contents of  $\mu_1^1$  are shifted along the lane to  $\mu_1^2$ , whilst the final result of the multiplication is stored in  $\mu_0^2$

i3 add R5,  $\mu_0^2$ ,  $\mu_1^2$ .

[0098] Furthermore, if the forwarding paths present in the microarchitecture so allow, transfers from microregisters in one lane to functional units in a different lane may be envisaged. In any case, the basic constraints for the compiler are - apart from the ones regarding latency - the following:

- the write microregister is always the one in the lane where the functional unit is located; forward transfers along the pipeline are also limited to the same lane;
- reading from microregisters is always allowed within the same lane and - in the case of different lanes - as far as forwarding paths make such reading possible.

[0099] Microregister use may become a liability in the event of interrupt handling, and, more in general, exception handling.

[0100] In fact, the microregisters may be regarded as constituting a "transient" memory such that could not be associated with a machine state to be saved in the case of an exception (except where a solution such as a shadow pipeline is envisaged).

[0101] As regards interrupt handling, two possible solutions may be proposed to overcome this problem.

[0102] One first solution is based upon the definition of an "atomic sequence", in the sense that the sequence of instructions using the microregisters is viewed as an atomic one and, as such, one that cannot be interrupted. Interrupt is disabled prior to start of the sequence, and the state of the machine is rendered stable (by writing in the RF or in the memory) before the interrupt is re-enabled. This solution does not require any extension of the instruction set or of the microarchitecture and is actually handled by the compiler alone.

[0103] Another solution is based upon a principle that may be referred to as "checkpointing".

[0104] Two new instructions (actually, pseudo-instructions used by the compiler and affecting only the control unit, but not the pipelines) are introduced, namely, checkpoint declaration (ckp.d) and checkpoint release (ckp.r).

[0105] At checkpoint declaration, the program counter (PC) is saved in a shadow register, and until checkpoint release the machine state cannot be modified (obviously, this implies that no storage instructions are allowed). At checkpoint release, the shadow register is reset, and the interrupts are disabled atomically. The results computed in the checkpointed section can be definitively stored (committed) so modifying the real state of the processor, after which the interrupts are enabled again to restart normal execution. In the case of an interrupt between ckp.d and ckp.r, the PC from which execution will restart after interrupt handling is the one saved in the shadow register (and, obviously, in view of the aforementioned constraints imposed on machine-state updating, the machine state is consistent with the said PC).

[0106] In this connection, two alternative solutions may be proposed.

[0107] According to the first solution, all register writes in the sequence between ckp.d and ckp.r involve only micro-registers. The compiler verifies whether there is a schedule satisfying the constraints imposed. The RF is involved only to read data.

[0108] According to the second solution, a (small) subset of the RF is reserved in a conventional way for "transient" variables between checkpoint declaration and checkpoint release, the liveness of which exceeds the maximum one allowed by the pipeline length. The first appearance of "transient" registers in the checkpointed sequence must be a definition (either a load or a write to a register). These transient registers are not seen as a constituent part of the machine state after checkpoint release (that is, they are considered dead values after this point). It should be noted that, obviously, adoption of these transient registers might imply the risk of register spilling. Quite simply, should register spilling become necessary, use of the microregisters is excluded, and normal compilation using the RF is adopted.

[0109] Of course, without prejudice to the principle of the invention, the details of construction and the embodiments may vary widely with respect to what is described and illustrated herein, without thereby departing from the scope of the present invention as defined in the attached claims.

5

## Claims

1. An for a pipeline processor circuit, comprising a plurality of stages (IF, ID, EX, MEM, WB) and a network of forwarding paths (EX-EX, MEM-EX, MEM-ID) which connect said stages, as well as a register file (RF) for operand write-back, **characterized in that** it comprises an optimization-of-power-consumption function via inhibition of writing (*Write inhibit*) and subsequent readings in said register file (RF) of operands retrievable from said forwarding network on account of their reduced liveness length.
2. An architecture according to Claim 1, **characterized in that** said function is configured for performing selectively, for at least one given register (R) assigned by a first instruction ( $w_d$ ) comprising a write-back stage (WB) and used by a second instruction ( $w_k$ ), the following:
  - disabling of write-back of said register (R) in said register file (RF) in the write-back stage (WB) of said first instruction; and
  - inhibiting assertion of the read address of said register (R) in said register file (RF) by said second instruction ( $w_k$ ).
3. An architecture according to Claim 1 or Claim 2, **characterized in that** it comprises a dedicated logic for disabling the write-enable signals that enable writing in said register file (RF).
4. An architecture according to any of the preceding claims, **characterized in that** it comprises a dedicated logic which minimizes the read-port switching activity in said register file (RF) by maintaining the values on the input read addresses at the previous clock cycles.
5. An architecture according to Claim 3 or Claim 4, **characterized in that** it comprises a decoding stage (ID) for decoding the instructions and reading the operands from said register file, and **in that** said dedicated logic is included in said stage (ID).
6. An architecture according to any of the preceding claims, **characterized in that** said processor is a superscalar processor comprising a hardware control unit capable of analyzing the instruction window to determine the liveness length of the registers.
7. An architecture according to any of Claims 1 to 5, **characterized in that** it is configured as a VLIW architecture, in which the decision of activating said inhibition function is delegated to the compiler.
8. An architecture according to Claim 7, **characterized in that** the compiler transfers the information to the hardware control logic, reserving specific operation bits in the instruction encoding.
9. An architecture according to Claim 7, **characterized in that** the compiler transfers the information to the hardware control logic, exploiting unused instruction encoding bits.
10. An architecture according to any of the preceding claims, **characterized in that** it includes interstage registers (EX/MEM, MEM/WB) comprised between the pipeline-structure stages for storing a volatile architectural state, and **in that** the architecture is configured for discarding elements that exit said volatile state, avoiding write-back in said register file (RF).
11. An architecture according to Claim 10, **characterized in that** it is adapted to operate on instructions configurable as exceptions, and **in that**, in order to ensure re-execution of instructions constituting an exception in the correct processor state, writing-back is envisaged of the values inhibited as regards writing in said register file (RF) in the presence of a signal that is configured as an exception.
12. An architecture according to Claim 11, **characterized in that** it comprises a decoding stage (ID) for decoding instructions and reading operands from said register file (RF), an instruction-execution stage (EX), and a memory-

access stage (MEM), and **in that** said write-back is envisaged whenever an exception signal is generated in one of said stages (ID, EX, MEM).

- 5        13. An architecture according to Claim 11, **characterized in that**, in the presence of an instruction configured as an exception, the architecture is configured for executing the instructions in the pipeline until their completion, there being envisaged write-back, in said register file (RF), of the results of all the instructions in the pipeline.
- 10      14. An architecture according to any of the preceding claims, **characterized in that** it comprises interstage registers, such as latch registers ( $\mu_0, \mu_1, \mu_2$ ) used as a memory layer for storing the operands.
- 15      15. An architecture according to Claim 14, **characterized in that** said interstage registers ( $\mu_0, \mu_1, \mu_2$ ) are configured in such a way that they are visible to the compiler and are not visible to the programmer.
- 20      16. An architecture according to Claim 14 or Claim 15, **characterized in that** said interstage registers ( $\mu_0, \mu_1, \mu_2$ ) are not write-addressable, in so far as they are implicitly addressed.
- 25      17. An architecture according to any of Claims 14-16, **characterized in that** said interstage registers ( $\mu_0, \mu_1, \mu_2$ ) are configured as a transient memory which cannot be associated to a machine state that can be saved in the event of an exception.
- 30      18. An architecture according to Claim 17, **characterized in that** it is configured in such a way that the sequences of instructions that use said interstage registers ( $\mu_0, \mu_1, \mu_2$ ) are treated as atomic sequences that are not subject to interrupts.
- 35      19. An architecture according to Claim 18, **characterized in that** disabling of any interrupt is envisaged prior to start of said sequences, and **in that** the machine state is rendered stable prior to interrupt re-enabling, for example by means of write-back in the register file or in the memory.
- 40      20. An architecture according to Claim 17, **characterized in that** it comprises a function of generation of two pseudo-instructions, one for checkpoint declaration (ckp.d) and one for checkpoint release (ckp.r), with the provision of a shadow register, wherein the program counter (PC) is saved from the instant of checkpoint declaration, the machine state not being modifiable until checkpoint release, whereby, upon checkpoint release, the shadow register is reset and the interrupts are disabled atomically.
- 45      21. An architecture according to Claim 20, **characterized in that** the results computed between said two pseudo-instructions are entrusted to the real state of the processor with subsequent interrupt re-enabling to enable re-start of normal execution.
- 50      22. An architecture according to Claim 20 or Claim 21, **characterized in that**, in the presence of interrupts between said pseudo-instructions, the execution is made to restart, after handling of the interrupts, starting from the program counter (PC) stored in the shadow register.
- 55      23. An architecture according to Claim 20, **characterized in that** all the register writings comprised between said pseudo-instructions involve only said interstage registers, whereby said register file (RF) is involved only for data reading.
- 60      24. An architecture according to Claim 20, **characterized in that** it comprises a subset of the register file (RF) reserved for the transient variables that are generated between said two pseudo-instructions and the liveness length of which exceeds the maximum value allowed by the pipeline.
- 65      25. An architecture according to Claim 24, **characterized in that** the first appearance of transient registers in the sequence being checkpointed is a definition such as a load or write in a register, which can be seen as a constituent part of the machine state after checkpoint release.

55

Fig. 1

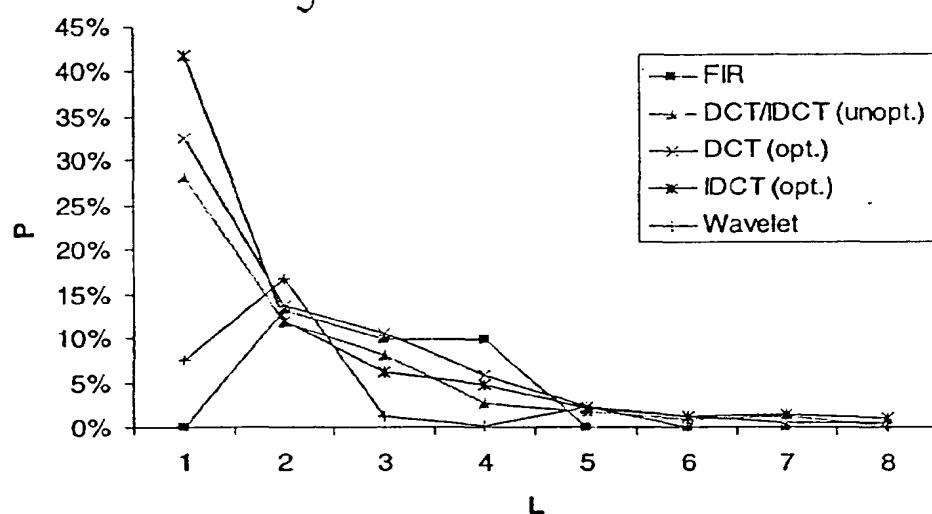
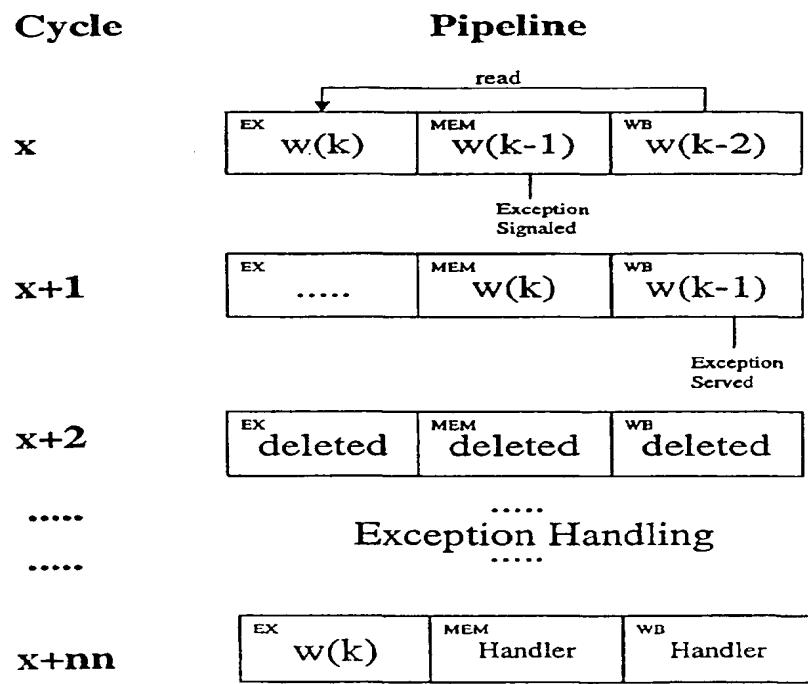


Fig. 3



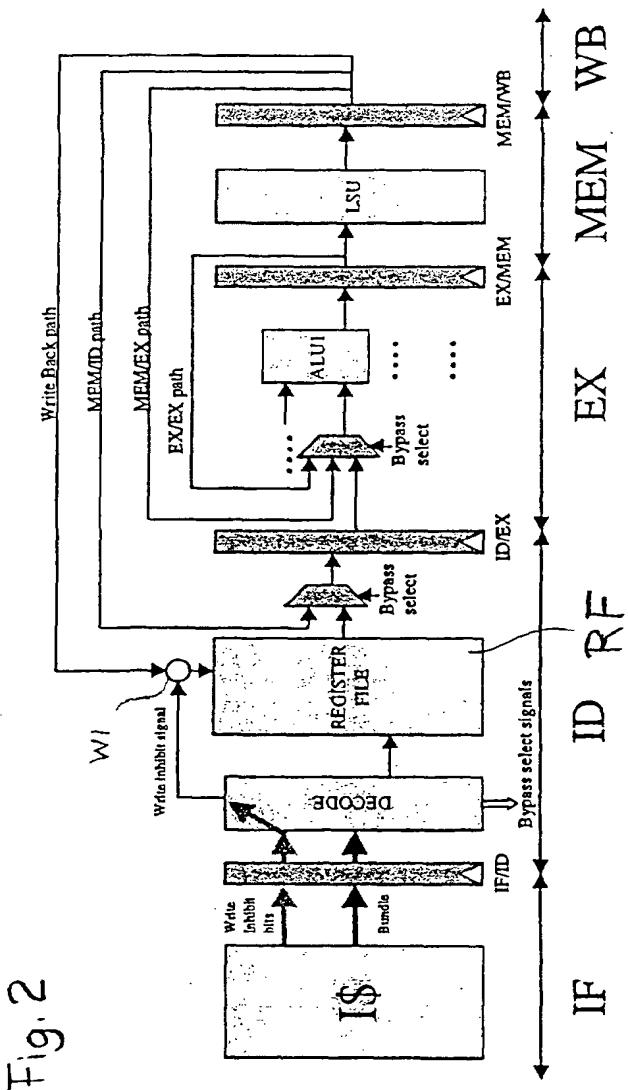
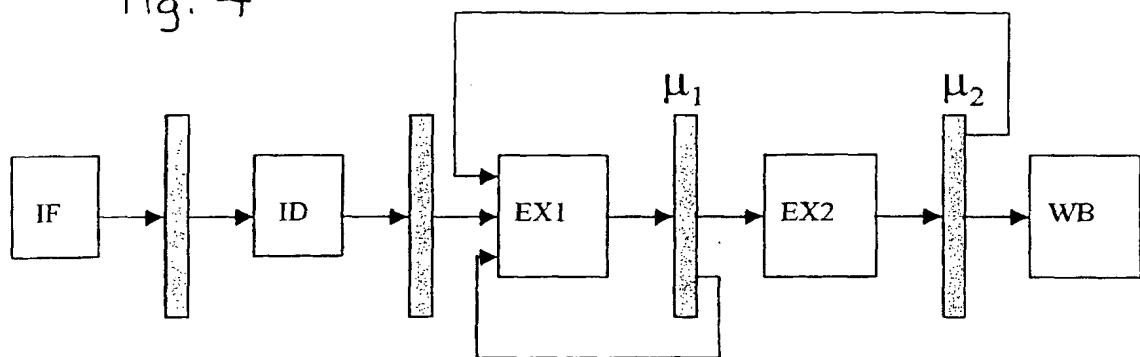


Fig. 4



RF

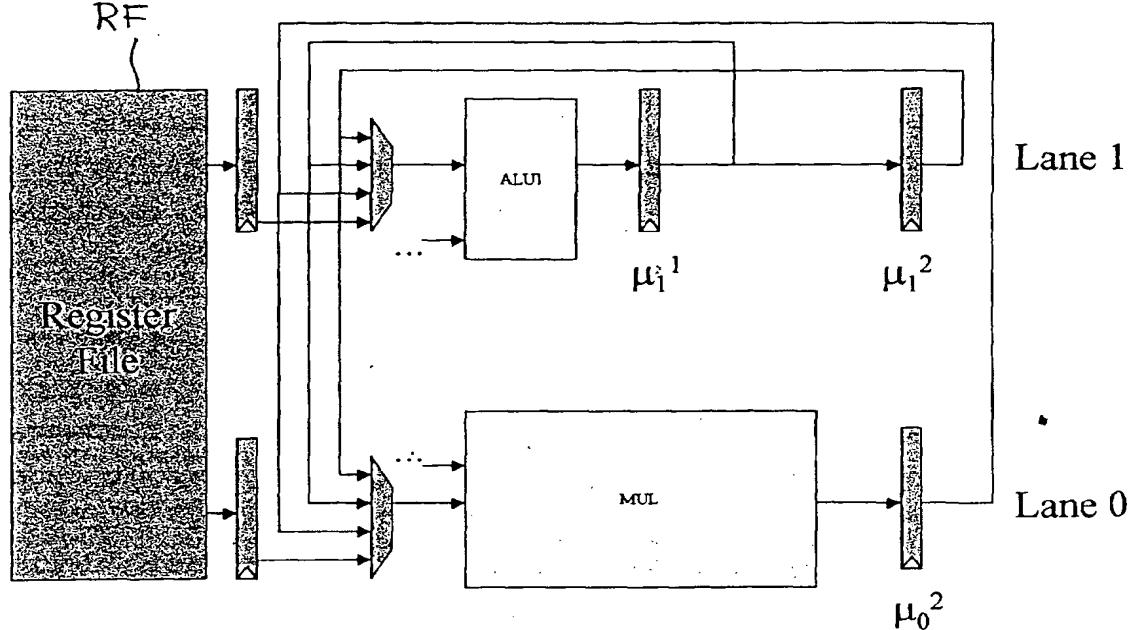


Fig. 5



European Patent  
Office

## EUROPEAN SEARCH REPORT

Application Number

EP 00 83 0673

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
D,A	<p>LOZANO C L A ET AL: "EXPLOITING SHORT-LIVED VARIABLES IN SUPERSCALAR PROCESSORS"            PROCEEDINGS OF THE ANNUAL INTERNATIONAL SYMPOSIUM ON MICROARCHITECTURE, US, LOS ALAMITOS, IEEE COMP. SOC. PRESS, vol. SYMP. 28, 29 November 1995 (1995-11-29), pages 292-302, XP000585373            ISBN: 0-8186-7349-4            * the whole document *</p> <p>---</p>	1-9	G06F9/38
A	<p>ARMIN LIEBCHEN ET AL: "DYNAMIC REORDERING OF HIGH LATENCY TRANSACTIONS USING A MODIFIED MICROPIPELINE"            PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON COMPUTER DESIGN: VLSI IN COMPUTERS AND PROCESSORS, US, LOS ALAMITOS, IEEE COMP. SOC. PRESS, vol. -, 11 October 1992 (1992-10-11), pages 336-340, XP000344638            ISBN: 0-8186-3110-4            * the whole document *</p> <p>---</p>	1	TECHNICAL FIELDS SEARCHED (Int.Cl.7)
A	US 4 228 497 A (GUPTA RAM K ET AL) 14 October 1980 (1980-10-14)	---	G06F
A	EP 0 569 312 A (IBM) 10 November 1993 (1993-11-10)	-----	
<p>The present search report has been drawn up for all claims</p>			
Place of search	Date of completion of the search	Examiner	
THE HAGUE	1 June 2001	Daskalakis, T	
CATEGORY OF CITED DOCUMENTS			
X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document		T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document	

**ANNEX TO THE EUROPEAN SEARCH REPORT  
ON EUROPEAN PATENT APPLICATION NO.**

EP 00 83 0673

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report.  
*The members are as contained in the European Patent Office EDP file on*  
 The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

01-06-2001

Patent document cited in search report		Publication date		Patent family member(s)	Publication date
US 4228497	A	14-10-1980		DE 2847575 A FR 2408174 A GB 2008819 A,B JP 1356447 C JP 54075935 A JP 61020013 B	23-05-1979 01-06-1979 06-06-1979 13-01-1987 18-06-1979 20-05-1986
EP 0569312	A	10-11-1993		US 5471626 A JP 2750311 B JP 6095872 A	28-11-1995 13-05-1998 08-04-1994

EPO FORM P14SC

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82